How To Fix The Old And Busted: The Science Behind Loaders

Whoami

- Been working in security about 10 years
- Author of some tools:
  - Scarecrow
  - Freeze/Freeze.rs
  - Mangle
  - Ivy
  - SourcePoint
- Numerous publications and interviews
- Lead of Optiv’s Red / Purple Team

Loadout

01 Define what a loader is and isn’t

02 Understand detection

03 Review my approach to continue analysis

04 Apply techniques to increase our success

Picture This

Grab your favorite Github tool

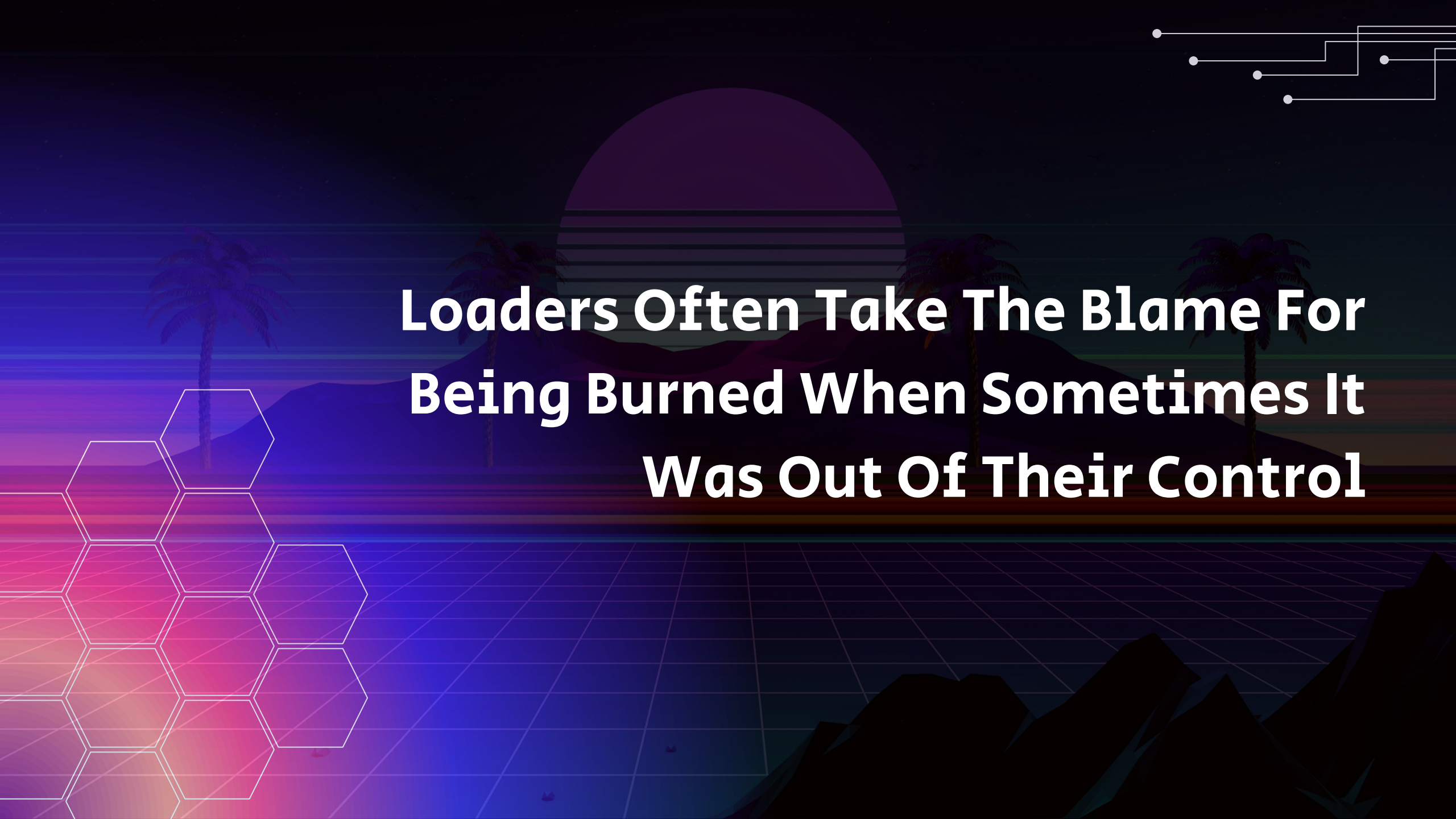
Build your payload with your favorite delivery method

Test it

Launch It

Alert

Burn the entire thing down and restart


Loaders Often Take The Blame For
Being Burned When Sometimes It
Was Out Of Their Control

Three Branches for Success

App or process blocked: PSEXESVC.exe

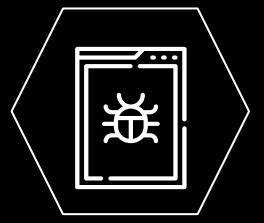
Blocked by: Attack surface reduction
Rule: Block process creations originating from PSExec and WMI commands

Affected items: C:\Windows\System32\cmd.exe

Branches

The Command

The command to deliver or execute the file to endpoint



The Loader

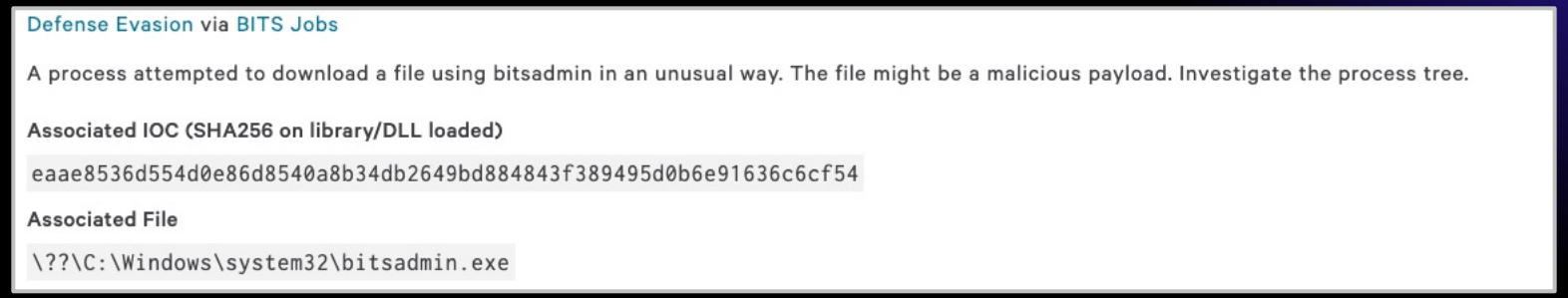
The file that contains shellcode and your evasion techniques

The Shellcode

The code used to call home and interact remotely with the endpoint

Why Not Just Commands ?

- **Known Behavioral Patterns**:
  - Many LOLBins have been abused by attackers in well-documented campaigns
  - Easy to detect and prevent the abuse of common LOLBins like **bitsadmin** and **regsvr32**
- **Increased Detection Rates**:
  - This has led to security solutions and detection engineers developing specific behavioral patterns and indicators of compromise (IoCs) to identify their malicious use
  - As a result, using these tools may trigger detection based on known malicious patterns



Example : Command line Detection

- DLL based loader file
- Executed with rundll32

- Rerun same loader but with Regsvr32

Why Not Just Shellcode?

DevSec

- You want multiple layers of defense
- At the mercy of the C2 author’s code base

OpSec Reasons

- Creates a single point of failure

A Loader's Responsibilities

Execution & Runtime Checks

Establish a process to perform the subsequent tasks and avoid static analysis

Perform Evasive Tactics

Take programmatic steps to ensure shellcode remains undetected

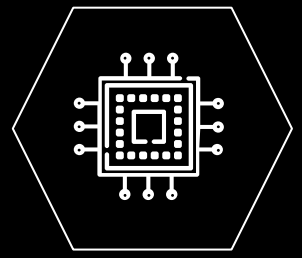
Execute Shellcode Safely

Establish the command and control channel

Loaders: Kryptonite

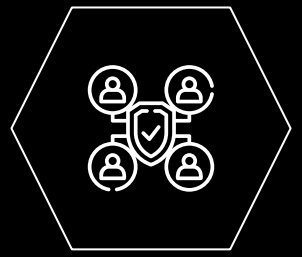
- **Application Controls**:
  - Allow you to specify which applications and processes are allowed to run on your system
- This means you prevent any unknown (subsequently never-before-seen) code from running
- Can’t really bypass these, but sidestepping or “sideloading” works well

Keys to be Successful



Detections != Straightforward

EDR algorithms are all private and trade secrets



Understand the Tools

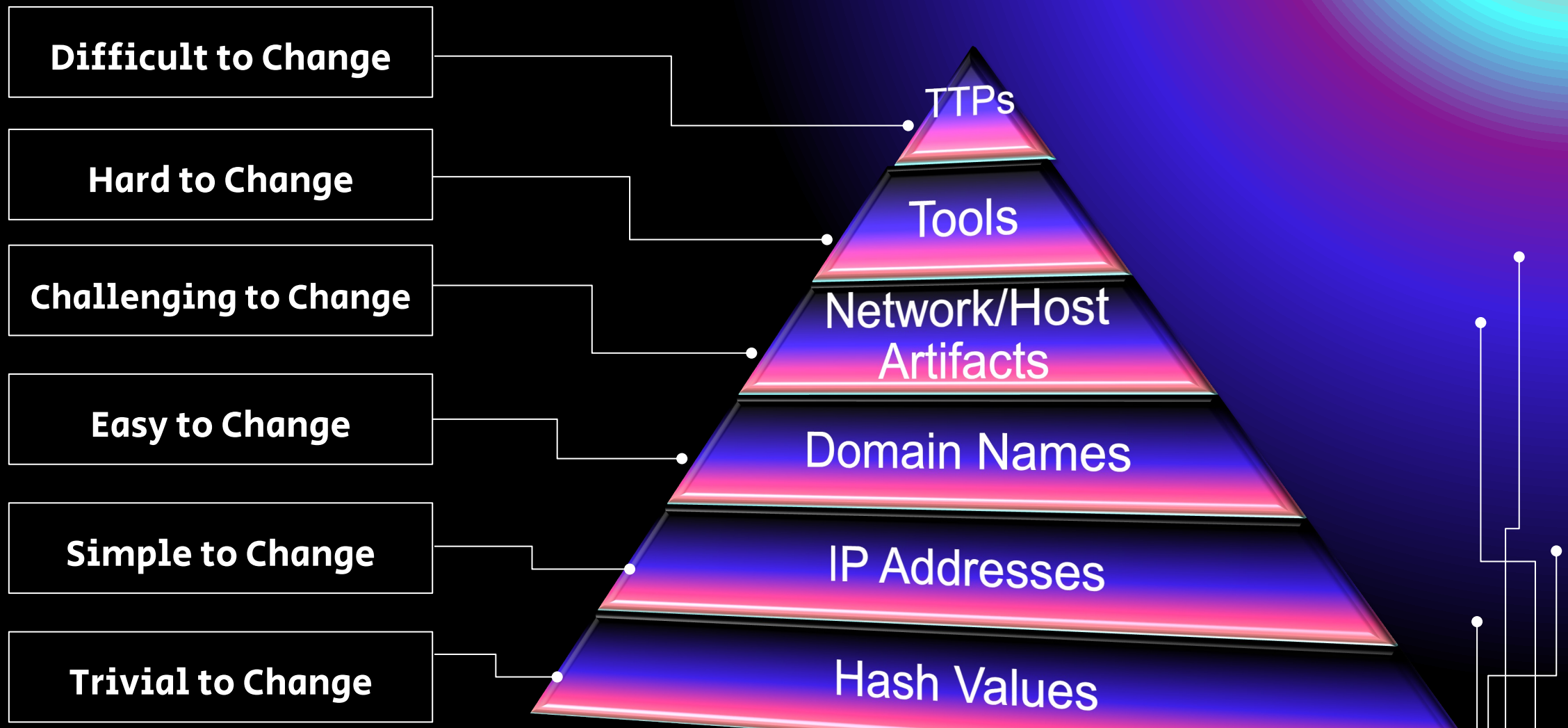
To improve our tradecraft, we must understand what is triggering the detection in the first place

Develop an Approach

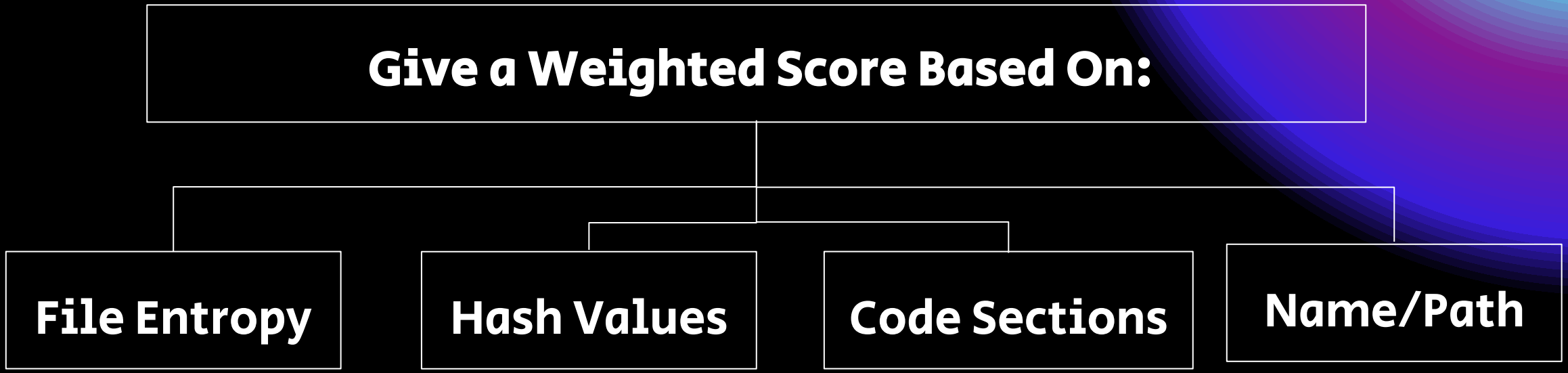
Need to approach them from a blind perspective – you never know what's running until you're there

Understanding Detection

Pyramid of Pain



EDR Machine Learning AV Engine Metrics



- They are combining lots of data points to see if it's bad
- Don't drive yourself crazy trying to understand what's under the hood

Detection Techniques

- These are given a weighted score based on confidence, and tries to map it to a specific malicious pattern set or "malware.gen" if it just thinks it's bad
- Someone once said “*SOC analysts don’t care how the alert was generated, just that an alert was generated*”
- Detection Engineering – Don’t look for technique, but rather find something to alert on that is systemic across all implants of it
- As SMEs, we should ask: why?

Defining The Situation

Situation

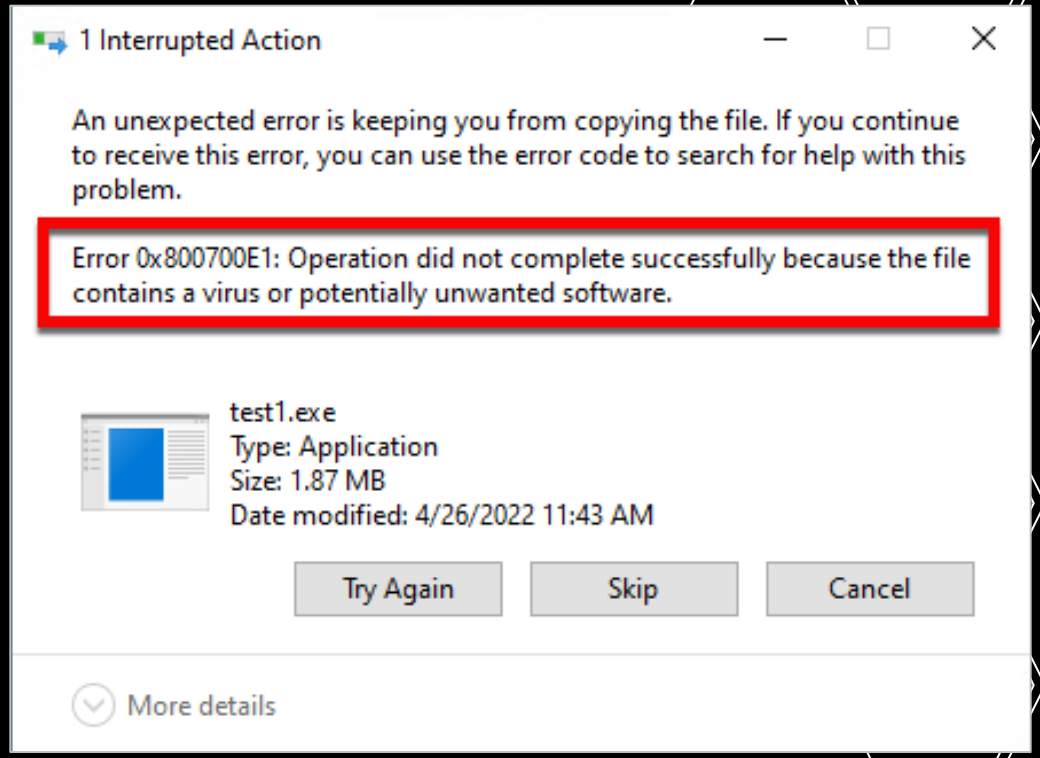
- If we have a sense of some common strings to avoid, we can just **strip them out of the file,** right?

Causes a PE mis-alignment

- An increase to the section size
- A decrease to the section size
- Changing a referenced function or library

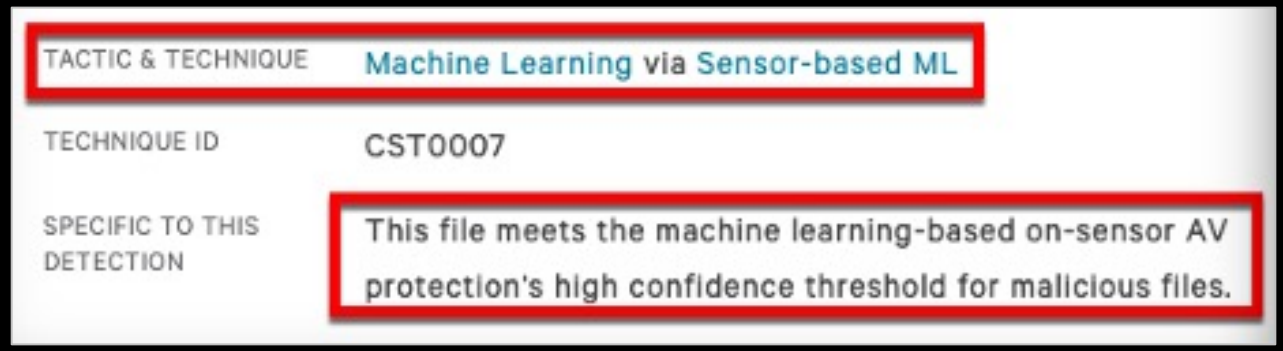
Situation:

- No Runtime events were performed:
  - EDR Unhooking
  - Module Stomping
  - Thread Spoofing
- Why did this happen?
  - The file contained something they could alert on



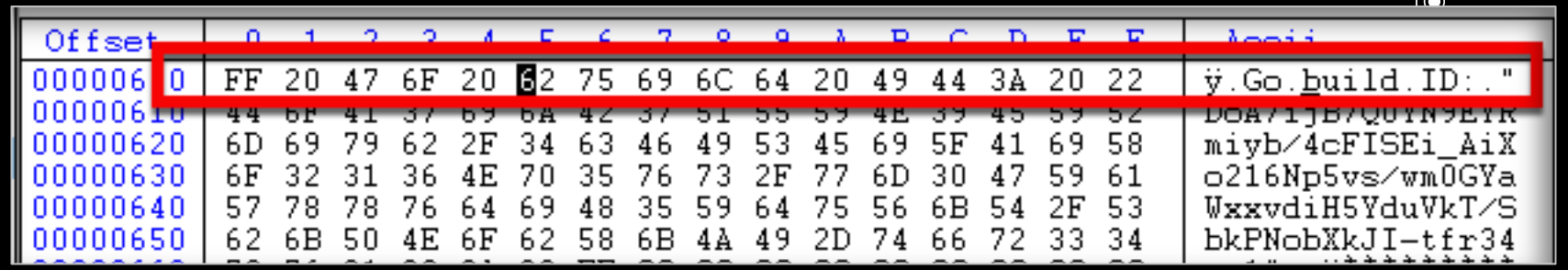
Reviewing the Logs

- Machine Learning is often an EDR's secret sauce
- So, we can't ever fully know what it's picking up on, and that's the problem
- So long as the alert is there, the product is doing its job
- Truth is, you don't know where the gaps are

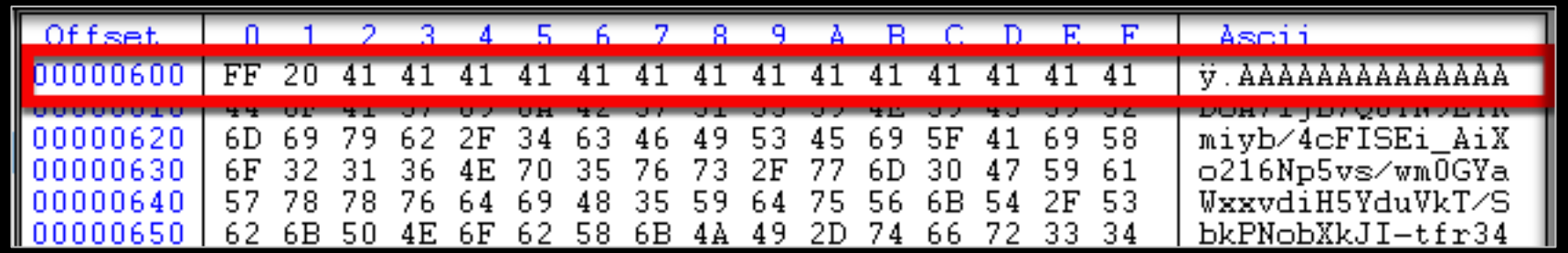


Concepts

- We must edit the EXACT bytes (nothing more, nothing less)
- We can’t change any function names



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
000006	0	FF	20	47	6F	20	62	75	69	6C	64	20	49	44	3A	20	22	ÿ.Go.build.ID:."
00000620		6D	69	79	62	2F	34	63	46	49	53	45	69	5F	41	69	58	miyb/4cFISEi_AiX
00000630		6F	32	31	36	4E	70	35	76	73	2F	77	6D	30	47	59	61	o216Np5vs/wm0GYa
00000640		57	78	78	76	64	69	48	35	59	64	75	56	6B	54	2F	53	WxxvdiH5YduVkT/S
00000650		62	6B	50	4E	6F	62	58	6B	4A	49	2D	74	66	72	33	34	bkPNobXkJI-tfr34



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000600	FF	20	41	41	41	41	41	41	41	41	41	41	41	41	41	41	ÿ.AAAAAAAAAAAAAA
00000620	6D	69	79	62	2F	34	63	46	49	53	45	69	5F	41	69	58	miyb/4cFISEi_AiX
00000630	6F	32	31	36	4E	70	35	76	73	2F	77	6D	30	47	59	61	o216Np5vs/wm0GYa
00000640	57	78	78	76	64	69	48	35	59	64	75	56	6B	54	2F	53	WxxvdiH5YduVkT/S
00000650	62	6B	50	4E	6F	62	58	6B	4A	49	2D	74	66	72	33	34	bkPNobXkJI-tfr34

Some Success

- When we test this, we see something different happen
- Rather than the Machine Learning flagging it before, we can run it now
- When we execute it, we see these beacons come in:

HAPUBWS-PC	0f8505baf9fcefee3749c4b73b85af7d472e0e1
HAPUBWS-PC	cbdaf29fcba21662e391eb553e8c41972e46948

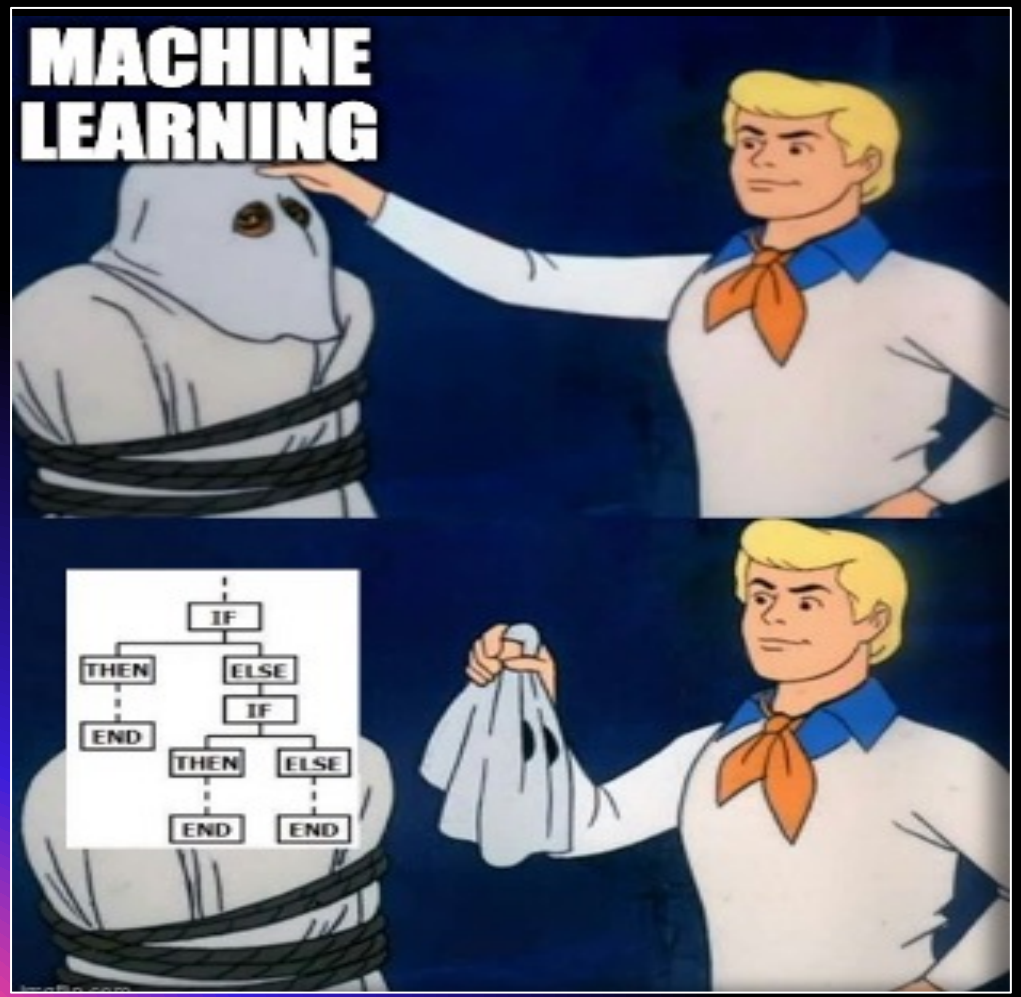
- Which is the Hash value of our test files:

```
meidelberg@OPT008385 Test Cases % shasum -a 256 code.exe
0f8505baf9fcefee3749c4b73b85af7d472e0e1d3f6f110442a8f348736410e3  code.exe
meidelberg@OPT008385 Test Cases % shasum -a 256 iMKrmHICwlkTmCF.exe
cbdaf29fcba21662e391eb553e8c41972e46948f43fcf14f3cb5e829bf1899d7  iMKrmHICwlkTmCF.exe
meidelberg@OPT008385 Test Cases %
```

What did we do?

- We removed enough of the static indicators on disk that it wasn't picked up
- This didn't mean that we're safe, but it does show that by removing these strings the EDR had to rely on other Telemetry (at Runtime) to see something suspicious.
- This also means that our other evasion techniques can now come into play:
  - **AMSI & ETW Patching**
  - **Unhooking EDR**
  - **Sandboxing**
  - **Module Stomping**
  - **Thread Spoofing**

EDR's Machine Learning – What it really is



- On-Disk scanning
- Looking for strings
- Pre-Run time scanning
- Checks behavioral patterns
  - Some even copy and execute in a container

Strings Manipulation

- Strings are often overlooked when we develop our attacks, but tools review them for:
  - Keywords
  - Malicious artifacts – URLs, IP addresses, Domains
  - File metadata - Strings in a file can provide information about the file's metadata
- Too much obfuscation can increase the files entropy (or using a packer)

```
meidelberg@OPT008385 Test Cases % strings StringTestCase.exe | grep .glob..| more
JZX93JG3.glob..func5
JZX93JG3.glob..func6
JZX93JG3.glob..func9
JZX93JG3.glob..func12
JZX93JG3.glob..func14
JZX93JG3.glob..func22
JZX93JG3.glob..func27
JZX93JG3.glob..func1
JZX93JG3.glob..func2
JZX93JG3.glob..func3
```

Pyramid Mod = Host Artifacts

Moving up the Pyramid

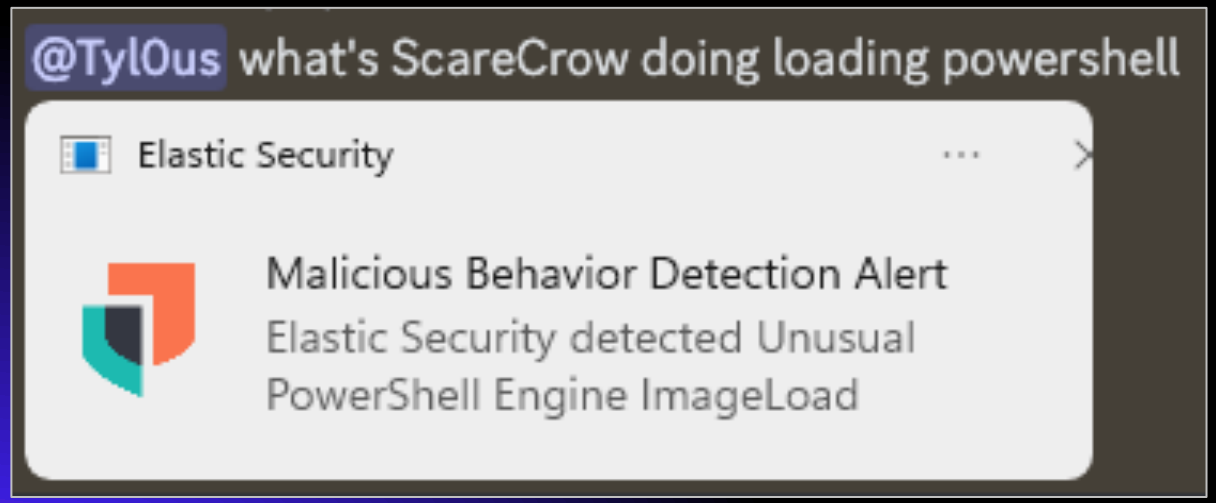
Some other easy wins

ETW Double Patching

- Event Tracing For Windows
- Great way to disable ETW telemetry
- Implementation:
  - Usually applied once - Error
  - If applied prior to any NTDLL unhooking, the patch is reversed
  - If applied after to any NTDLL unhooking, the events related to unhooking NTDLL will still be logged

Pyramid Mod = TTP

AAAA – Almost Always Avoid AMSI



- AMSI patching can get you caught more than it can help
- Why? Not all EDRs load AMSI into the process, which means the act of it being loaded to be patched is suspicious
- Loading AMSI can cause False Negatives

Pyramid Mod = TTP

EDR Unhooking

- EDR unhooking is a technique used to remove an EDR agent's hooks from the process's memory
- Everything in the userland space has the same permissions, therefore an attacker's code has the same permissions as an EDR
- Most effective when restoring the entire DLL(s), rather than a few bytes
- Common techniques to accomplish this:
  - Copying the clean version of hooked .dll from disk
  - Mapping and copying the .dll from the KnownDLL region of memory
  - Creating a suspended process

Pyramid Mod = TTP

Shellcode: Obfuscation != Encryption

- Obfuscation alters the presentation or representation of the data in a way that makes it harder to understand
- Encryption is a process that transforms data into an unreadable form, known as ciphertext, using a mathematical algorithm and a **secret key**
- A key is something unknown to security products so they can't account for it

Pyramid Mod = Host Artifacts

Avoid using VirtualProtect

- VirtualProtect changes the permissions on a defined region in memory, making something writeable that previously wasn't
  - Allocating memory with RWX or flipping permissions from RW to RX with VirtualProtect is suspicious to most EDRs

- WriteProcessMemory – Writes allocated buffer to specified region
  - Instead, allocate memory as RX, then use WriteProcessMemory to avoid problematic memory permissions.

Pyramid Mod – TTP

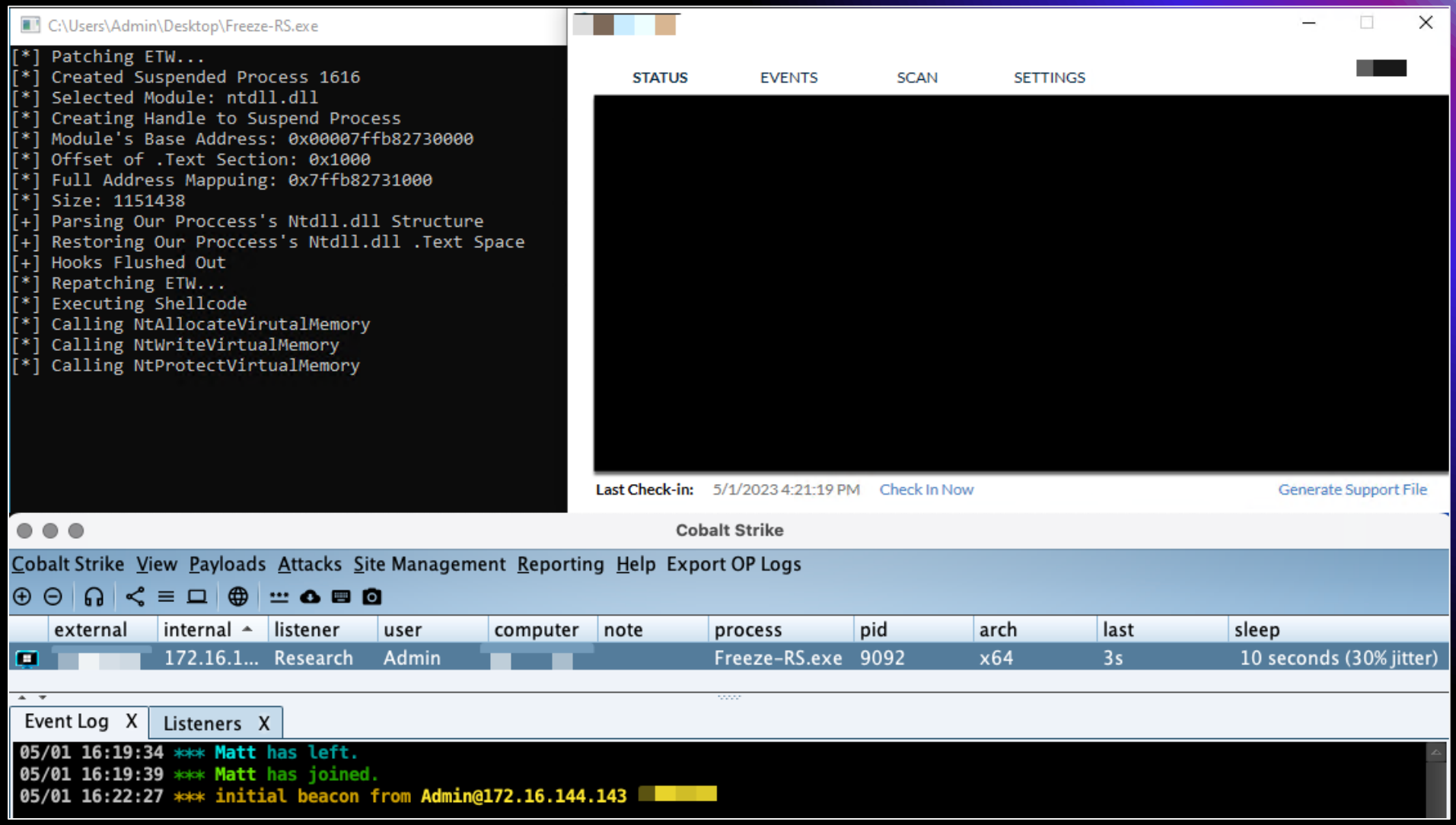
Sign Your Damn Code

- Most legitimate executables are signed by the vendor that published it
- By signing our code, we can blend in with the legitimate modules loaded into a process

Pyramid Mod = Tool

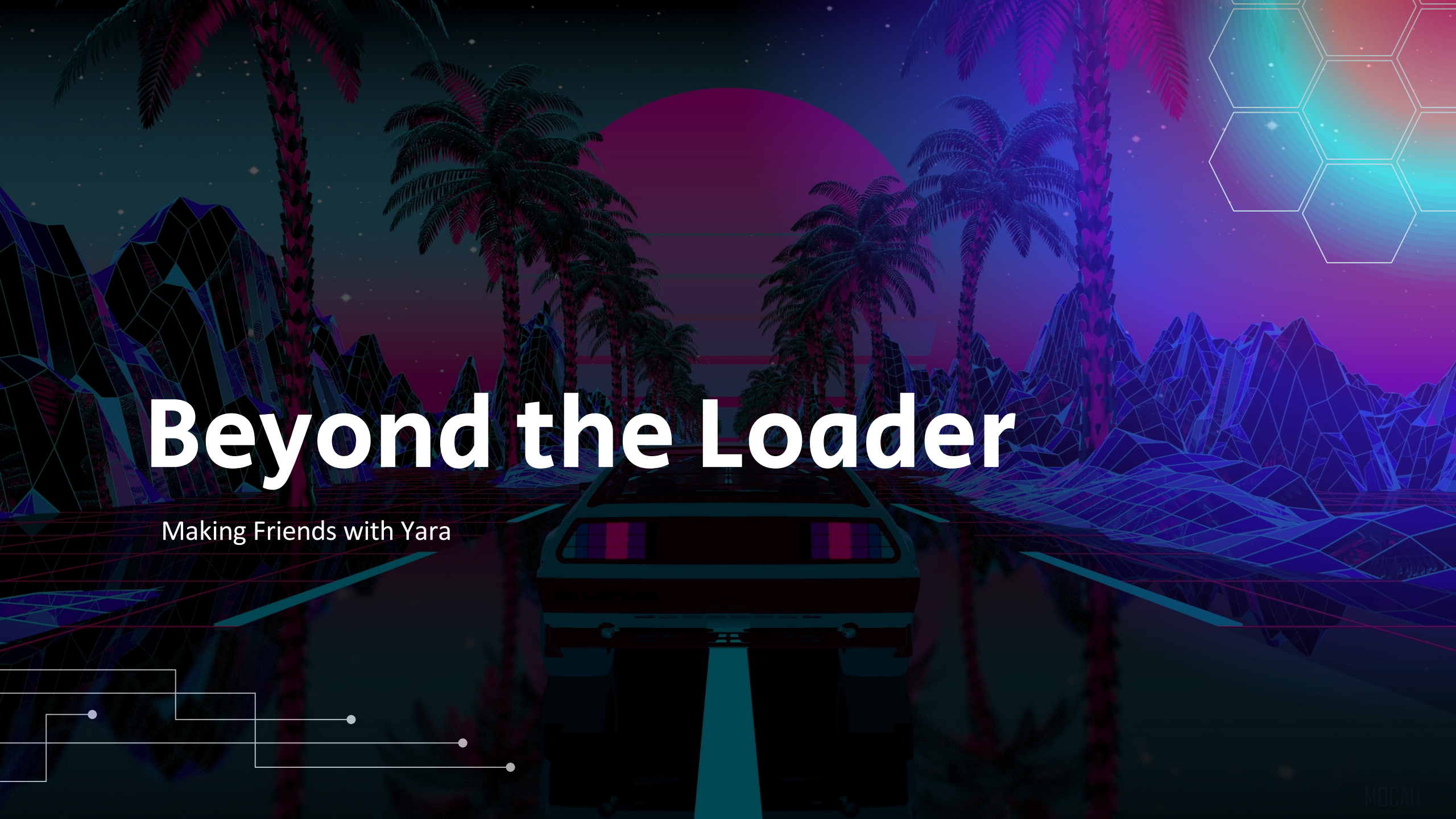
modload	Loaded c:\windows\system32\ctiuser.dll **Signed** (f2646843bb6e5a1e32f96724e1bc47d0)
modload	Loaded c:\windows\system32\fitlib.dll **Signed** (e202dd92848c5103c9abf8ecd22bc539)
modload	Loaded c:\windows\system32\uxtheme.dll **Signed** (3c9d22cae173ad19806b6a016cd4cc28)
crossproc	Opened handle with change access rights to c:\windows\system32\svchost.exe (9520a99e77d6
modload	Loaded c:\windows\system32\apphelp.dll **Signed** (929afbf6453da8e969c7bba44bce0906)
modload	Loaded c:\users\admin\desktop\rundll.cpl **Unsigned** (7bfd833ebd3c361f5e87ee8e58ad7a93)
modload	Loaded c:\windows\system32\mscoree.dll **Signed** (8ffaa80338156fd443d34f4ce9bd8431)

Combining These Things Together



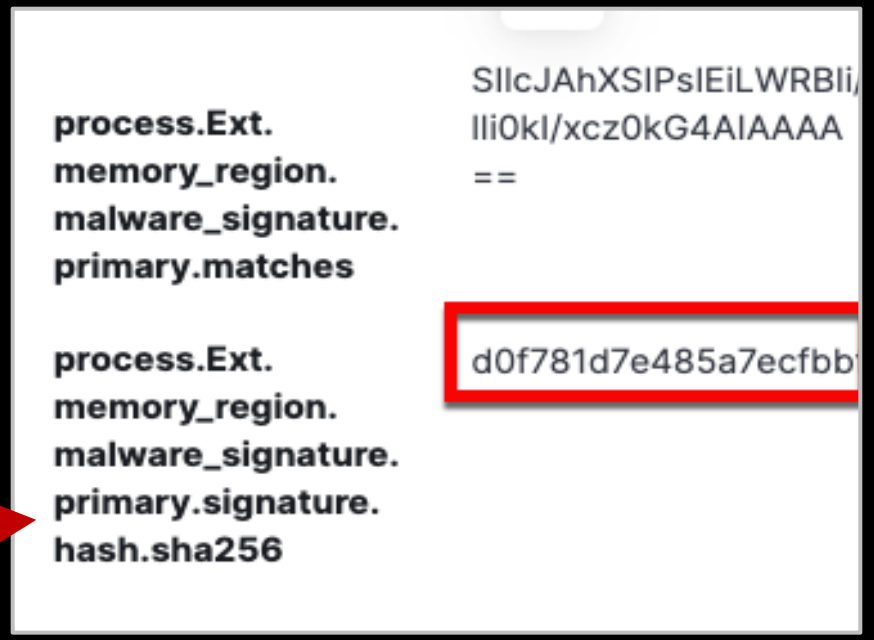
Pain Pyramid Modification Summary

- String Obfuscation – Host Artifacts
- ETW PATCH Double Patching – TTP
- Avoiding AMSI – TTP
- EDR unhooking – TTP
- Shellcode Encryption – Host Artifacts
- Writing Shellcode – TTP
- Code Signing - Tool


Beyond the Loader
Making Friends with Yara

Example: Beacon calls home, then dies

- EDR vendors don’t want to divulge their secret sauce
- But sometimes we can get lucky
- Looking at this alert, it refers to a memory section-based IOC



Digging Deeper

- A simple Google search of this string yields a JSON database of this EDR's Yara rules

- We can see a string of bytes that it is looking for:

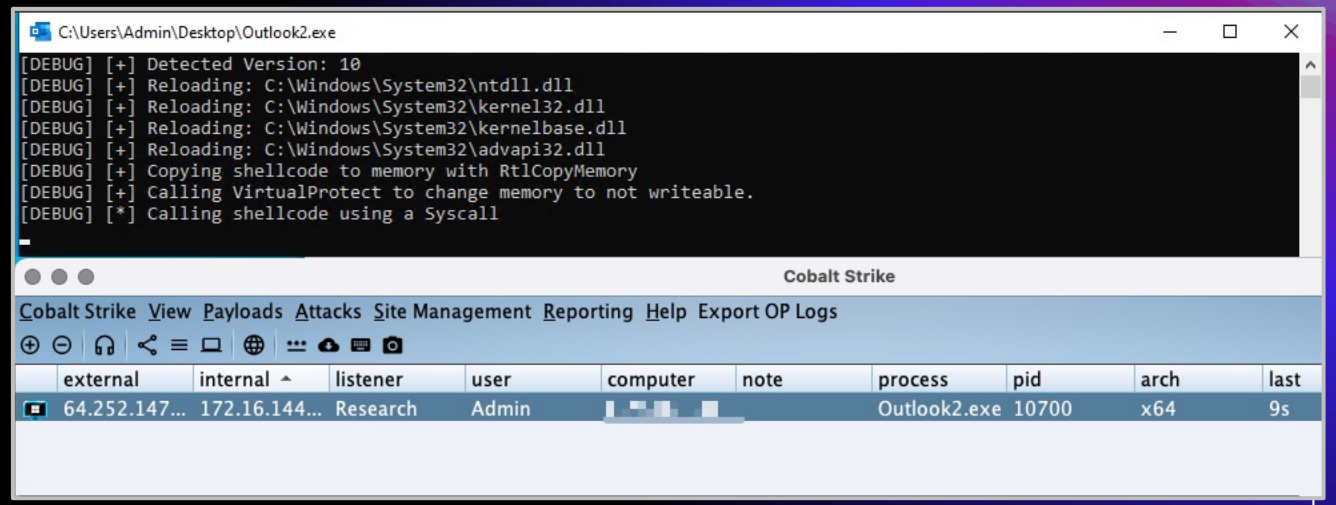
```
"yara_rule_data": "rule Windows_Trojan_Cobaltstrike_663fc95d { meta:  id = \"663fc95d-2472-4d52-ad75-c5d86cfc885f\"
    fingerprint = \"d0f781d7e485a7ecfbbf[illegible]068601e72430d57ef80fc92a993033deb1ddcee5c48\"  creation_date = \"2021-04-01\"
    last_modified = \"2021-04-01\"  description = \"Identifies CobaltStrike via unidentified function code\"  os = \"
    Windows\"  arch = \"x86\"  category_type = \"Trojan\"  family = \"Cobaltstrike\"  threat_name = \"
    Windows.Trojan.CobaltStrike\"  source = \"Manual\"  maturity = \"Diagnostic\"  scan_type = \"File, Memory\"  severity
    = 100 strings:  $a = { 48 89 5C 24 08 57 48 83 EC 20 48 8B 59 10 48 8B F9 48 8B 49 08 FF 17 33 D2 41 B8 00 80 00 00 }
    condition:  all of them}",
"scan_context": [
    "file",
    "memory"
],
```

Another Google Search

```
rule CobaltStrike_Sleep_Decoder_Indicator {
        meta:
                description = "Detects CobaltStrike sleep_mask decoder"
                author = "yara@s3c.za.net"
                date = "2021-07-19"
        strings:
                $sleep_decoder = { 48 89 5C 24 08 48 89 6C 24 10 48 89 74 24 18 57 48 83 EC 20 4C 8B 51 08 41 8B F0 48 8B EA 48 8B D9 45 8B 0A 45 8B 5A 04 4D
8D 52 08 45 85 C9 }
        condition:
                $sleep_decoder
}
```

Success

- Since we know what that alert was triggering on, we can adapt our tool
- A simple modification of code related to that string in memory circumvents the detection rule
- Executing an updated version yields a successful beacon that continuously calls home




How Can You Apply
these Techniques

Research Box

- Visual Studio code or studio
  - Developing and compiling your code
- Some sort of debugger
  - X64dbg - Great for debugging your code
- CFF Explorer
  - Great for reviewing the static strings
  - PE structure
  - Export Functions
- APIMonitor
  - Great for reviewing all the events occurring during runtime, including the values in each API call
- PeStudio
  - Let's you look at your files the same way many malware analysts were trained
  - Also great for reviewing the static strings and PE structure
- SysInternals Process Monitor

Takeaways

- Understanding/analyzing your entire attack chain is KEY
  - Understanding what's triggering an alert is very important
  - It may not always be your loader
- Evasion is not about completely changing up your TTPs because they caught a loader
  - Simple manipulations to the file can yield high levels of success
- There are many effective methods which, when combined, can yield high success
  - Loading them ALL can sometimes lead to the reverse effect
- There is no skeleton key for initial access:
  - Continuously tweak your tooling and approach
  - Something that works one week may not work the next week

Questions

Slides: https://github.com/Tylous/Slides

Reach out to me Twitter: Tyl0us

Thank you for listening to my talk